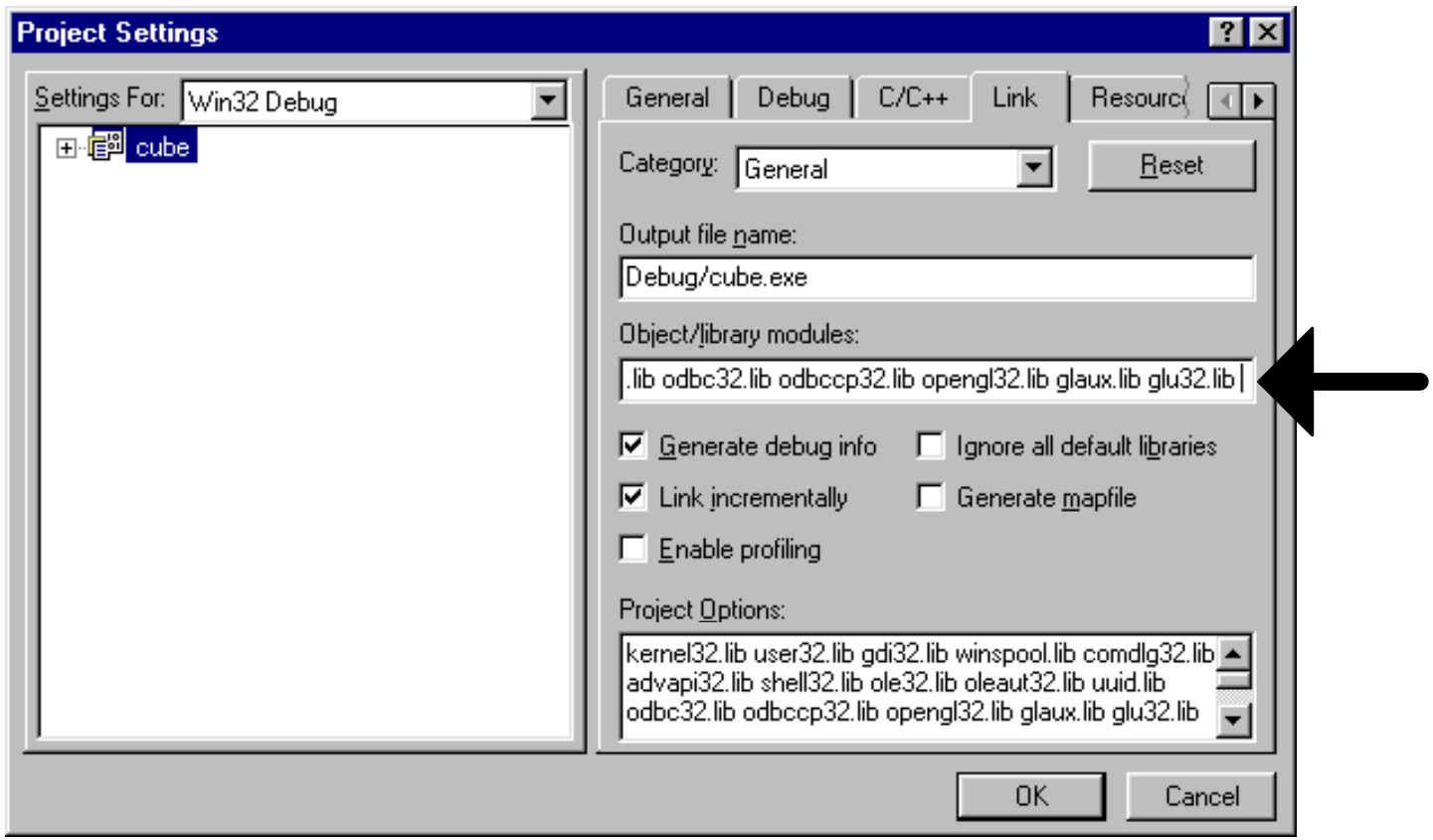
OpenGL - charakterystyka

- **OpenGL jest interfejsem programowym aplikacji – zestawem funkcji umożliwiających tworzenie interaktywnej grafiki 3D.**
- **Program oparty na OpenGL musi być pisany z zastosowaniem języka programowania (C, Visual Basic, Delphi itp.).**
- **Zestaw funkcji OpenGL podzielono na 3 podstawowe składniki:**
  - **Bibliotekę AUX (nowa wersja nosi nazwę GLUT)**
    - **Narzędzie do uruchamiania aplikacji OpenGL na dowolnej platformie systemowej (nie należy do standardu).**
  - **Bibliotekę GL**
    - **Zestaw funkcji zdefiniowanych w standardzie OpenGL.**
  - **Bibliotekę GLU**
    - **Zestaw funkcji wyższego poziomu umożliwiających rysowanie, oświetlenie i teksturowanie sfer dydków i walców, korzystających z biblioteki GL.**
- **Biblioteka OpenGL jest włączona jako składnik systemów operacyjnych Windows (od Win95 OSR), Unix oraz Linux.**

Modyfikacje konieczne w ustawieniach projektu Visual C++

- **Project->Settings, zakładka Link**
- **Należy uzupełnić listę bibliotek o: glaux.lib glu32.lib opengl32.lib**



Biblioteka AUX

- **Program z zastosowaniem biblioteki AUX (GLUT) tworzy się jako aplikację konsoli (typ projektu – Win Console Application, typowy program w języku C z główną funkcją main()).**
- **Podstawowy szablon aplikacji z zastosowaniem biblioteki AUX może mieć postać:**

```
#include <windows.h>
#include <gl\gl.h>
#include <gl\glaux.h>

void CALLBACK ChangeSize(GLsizei w, GLsizei h) {...}
void CALLBACK RenderScene(void)  {...}
void CALLBACK IdleFunction(void) {...   RenderScene();}
void main(void)
{   auxInitDisplayMode(AUX_DOUBLE | AUX_RGBA);
    auxInitPosition(100,100,250,250);
    auxInitWindow("Double Buffered Animation");
    auxReshapeFunc(ChangeSize);
    auxIdleFunc(IdleFunction);
    auxMainLoop(RenderScene);}
```

Biblioteka AUX – przykładowa funkcja IdleFunction

```
void CALLBACK IdleFunction(void)
    {
    // Modyfikuj położenie obiektu na scenie
    if(x1 > windowWidth-rsize || x1 < 0)
        xstep = -xstep;
    if(y1 > windowHeight-rsize || y1 < 0)
        ystep = -ystep;
    if(x1 > windowWidth-rsize)
        x1 = windowWidth-rsize-1;
    if(y1 > windowHeight-rsize)
        y1 = windowHeight-rsize-1;
    x1 += xstep;
    y1 += ystep;

    // Odrysuj scenę z nowymi koordynatami kwadratu
    RenderScene();
    }
```

Biblioteka AUX – przykładowa funkcja RenderScene

```
void CALLBACK RenderScene(void)
    {
    // Ustal kolor tła na niebieski:
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f);

    // Wyczyść okno przypomocy ustalonego koloru:
    glClear(GL_COLOR_BUFFER_BIT);

    // Ustal kolor malowanych obiektów na czerwony i rysuj
    // kwadrat w ustalonej bieżącej pozycji:
    glColor3f(1.0f, 0.0f, 0.0f);
    glRectf(x1, y1, x1+rsize, y1+rsize);
    // Opróżnij wszystkie bufory OpenGL:
    glFlush();

    // Zawartość namalowanego obrazu prześlij na ekran:
    auxSwapBuffers();
    }
```

Aplikacje OpenGL zgodne z Win32 - kontekst renderowania

```
LRESULT CALLBACK WndProc(  HWND   hWnd, UINT message,
                           WPARAM wParam, PARAM lParam )
{   static HGLRC hRC;                  // Kontekst renderowania
    static HDC hDC;                    // Kontekst urządzenia
    switch (message)
    {   case WM_CREATE:
              hDC = GetDC(hWnd);
              SetDCPixelFormat(hDC);      // format pikseli
              hRC = wglCreateContext(hDC);
              wglMakeCurrent(hDC, hRC);   //...
              break;
        case WM_DESTROY:
              wglMakeCurrent(hDC,NULL);
              wglDeleteContext(hRC);
              PostQuitMessage(0);
              break;
        // ...
     }     return (0L); }
```

Aplikacje OpenGL zgodne z Win32 – komunikat WM_PAINT

```
case WM_PAINT:
                {
                // Wywołaj funkcję rysującą
                RenderScene();

                // wywołaj funkcję zamieniającą bufory
                SwapBuffers(hDC);

                //Zabezpieczenie przed niepotrzebnymi wywołaniami
                // komunikatu WM_PAINT
                ValidateRect(hWnd,NULL);
                }
                break;
```

Aplikacje OpenGL zgodne z Win32 – tworzenie okna

```
hWnd = CreateWindow
(
        lpszAppName,
        lpszAppName,
        // OpenGL wymaga WS_CLIPCHILDREN i WS_CLIPSIBLINGS
        WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN | WS_CLIPSIBLINGS,
        // Pozycja i rozmiar okna
        100, 100,
        250, 250,
        NULL,
        NULL,
        hInstance,
        NULL
);
```

Aplikacje OpenGL zgodne z Win32 – format pikseli

```
void SetDCPixelFormat(HDC hDC)
{   int nPixelFormat;
    static PIXELFORMATDESCRIPTOR pfd = {
    sizeof(PIXELFORMATDESCRIPTOR),// Rozmiar struktury
    1,                            // Wersja struktury
    PFD_DRAW_TO_WINDOW |    // Rysuj w oknie nie w bitmapie
    PFD_SUPPORT_OPENGL |    // Wspomagaj OpenGL
    PFD_DOUBLEBUFFER,       // Tryb podwójnego buforowania
    PFD_TYPE_RGBA,          // Tryb kolorów RGBA
    8,                      // 8-bitowy kolor
    0,0,0,0,0,0,0,0,0,0,0,0, // Nie używane
    16,                     // Rozmiar bufora głębokości
    0,0,                    // Nie używane do wybrania trybu
    PFD_MAIN_PLANE,         // Rysuj w głównym planie
    0,0,0,0 };              // Nie używane do wybrania trybu

    nPixelFormat = ChoosePixelFormat(hDC, &pfd);
    SetPixelFormat(hDC, nPixelFormat, &pfd);    }
```

Aplikacje OpenGL zgodne z Win32 – zegar synchronizujący animację

```
case WM_CREATE:
        // ...
        // Utwórz Timer odpalający co 1 milisekundę
        SetTimer(hWnd,101,1,NULL);
        break;

case WM_DESTROY:
        // Zniszcz Timer
        KillTimer(hWnd,101);
        // ...
        break;

case WM_TIMER:
        {
        IdleFunction();
        InvalidateRect(hWnd,NULL,FALSE);
        }
        break;
```

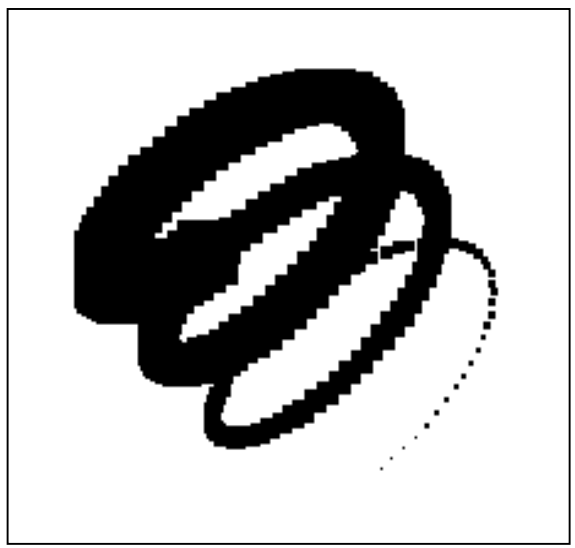
Prymitywy graficzne OpenGL - punkty

```
// Wyrysowanie dwu punktów:
glBegin(GL_POINTS);
    glVertex3f(0.0f,0.0f,0.0f);
    glVertex3f(50.0f,50.0f,50.0f);
glEnd();

// Definiowanie rozmiaru punktów:
{   GLfloat sizes[2];
    GLfloat step;

    glGetFloatv(GL_POINT_SIZE_RANGE,sizes);
    glGetFloatv(GL_POINT_SIZE_GRANULARITY,&step);

    glPointSize(GLfloat size); //ustal rozmiar punktu
}
// „Wygładzanie” punktów:
glEnable(GL_PIONT_SMOOTH);
```

GL_POINTS

Prymitywy graficzne OpenGL – punkty - przykład

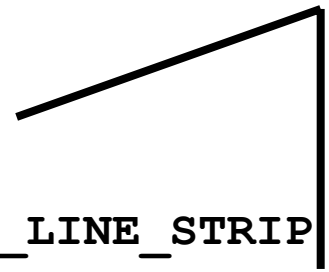
```
void RenderScene(void)
{   GLfloat x,y,z,angle, step, curSize;
    GLfloat sizes[2];
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0f, 0.0f, 0.0f);
    glPushMatrix();
    glGetFloatv(GL_POINT_SIZE_RANGE,sizes);
    glGetFloatv(GL_POINT_SIZE_GRANULARITY,&step);
    curSize = sizes[0];     z = -50.0f;
    for(angle=0.0f; angle<=(2.0f*3.1415f)*3.0f; angle+=0.1f)
    {   x = 50.0f*sin(angle); y = 50.0f*cos(angle);
        glPointSize(curSize);
        glBegin(GL_POINTS);
               glVertex3f(x, y, z);
        glEnd();
        z += 0.5f;
        curSize += step;     }
    glPopMatrix();   glFlush();}
```



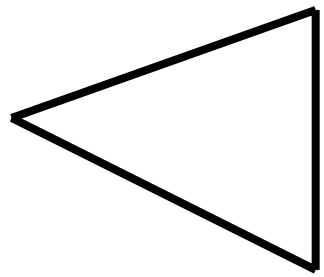
Prymitywy graficzne OpenGL - linie

```
// Pojedyncze linie
glBegin(GL_LINES);
    glVertex3f(0.0f,0.0f,0.0f);
    glVertex3f(50.0f,50.0f,50.0f);
glEnd();
// Łańcuch linii
glBegin(GL_LINE_STRIP); // glBegin(GL_LINE_LOOP);
    glVertex3f(0.0f, 0.0f, 0.0f);          //V0
    glVertex3f(50.0f, 50.0f, 0.0f);        //V1
    glVertex3f(50.0f, 100.0f, 0.0f);       //V2
glEnd();
// Definiowanie grubości linii:
{GLfloat sizes[2];
GLfloat step;
glGetFloatv(GL_LINE_WIDTH_RANGE,sizes);
glGetFloatv(GL_LINE_WIDTH_GRANULARITY,&step);

glLineWidth(GLfloat size[1]);}
```

GL_LINES



GL_LINE_STRIP



GL_LINE_LOOP

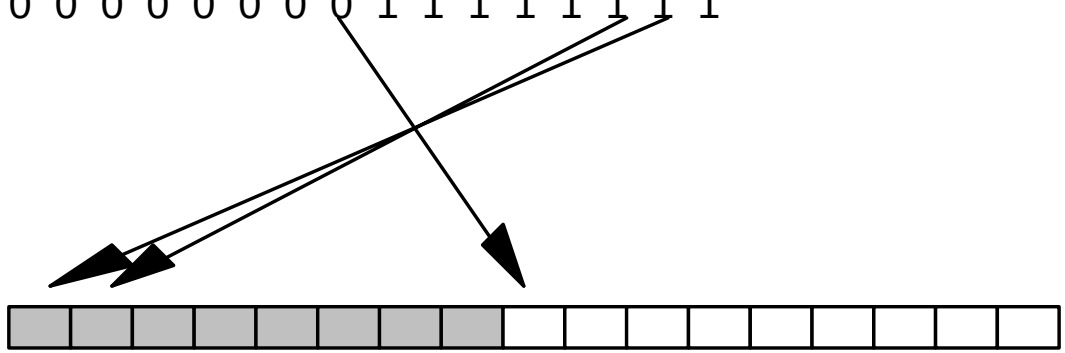
Prymitywy graficzne OpenGL – linie - wzory

```
glEnable(GL_LINE_STIPPLE);
```

```
void glLineStipple(GLint factor, GLushort pattern);
```

Pattern=0x00FF=255

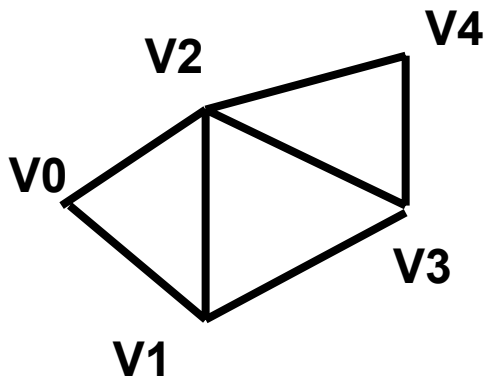
Binarnie: 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1



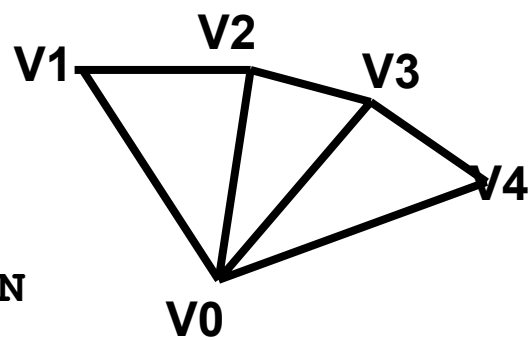
Prymitywy graficzne OpenGL – trójkąty

```
// Pojedyncze trójkąty
glBegin(GL_TRIANGLES);
    glVertex2f(0.0f, 0.0f);          //V0
    glVertex2f(25.0f, 25.0f);        //V1
    glVertex2f(50.0f, 0.0f);         //V2
    glVertex2f(-50.0f, 0.0f);        //V3
    glVertex2f(-75.0f, 50.0f);       //V4
    glVertex2f(-25.0f, 0.0f);        //V4
glEnd();
// Łańcuch trójkątów
glBegin(GL_TRIANGLE_STRIP);
    //...
glEnd();
// Wachlarz trójkątów
// glBegin(GL_TRIANGLE_FAN);
    //...
glEnd();
```

GL_TRIANGLES



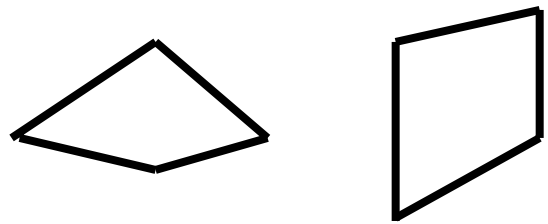
GL_TRIANGLE_STRIP



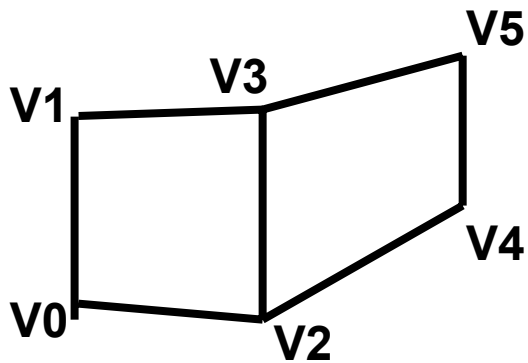
GL_TRIANGLE_FAN

Prymitywy graficzne OpenGL – czworokąty

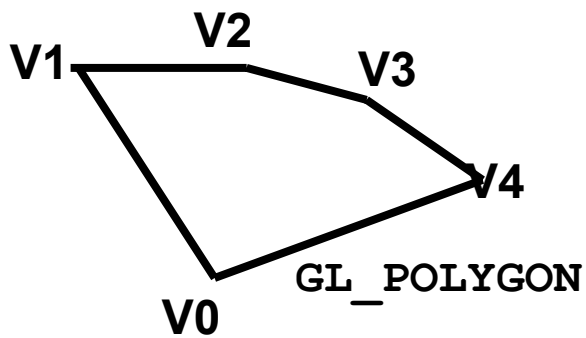
```
// Pojedyncze czworokąty
glBegin(GL_QUADS);
    glVertex2f(0.0f, 0.0f);        //V0
    glVertex2f(25.0f, 25.0f);      //V1
    glVertex2f(50.0f, 0.0f);       //V2
    glVertex2f(-50.0f, 0.0f);      //V3
    glEnd();
// Łańcuch czworokątów
glBegin(GL_QUAD_STRIP);
    //...
glEnd();
// Wielokąty
glBegin(GL_POLYGON);
    //...
glEnd();
```



GL_QUADS



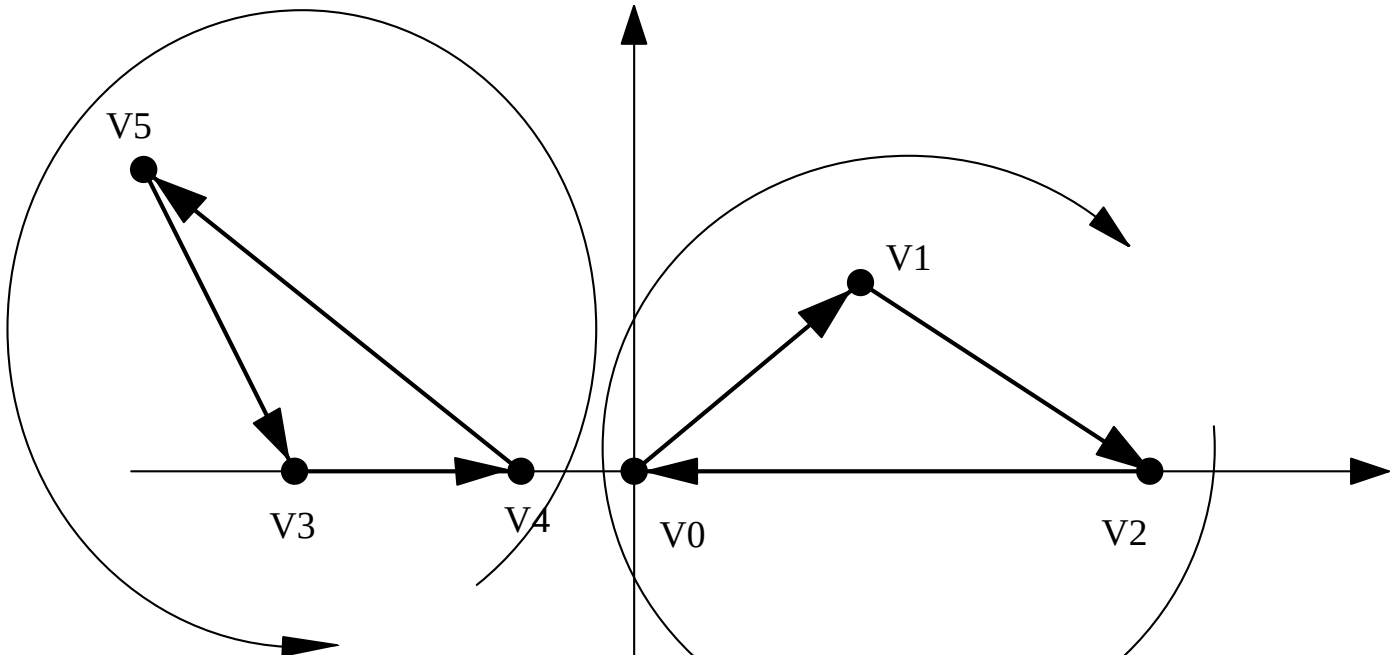
GL_QUAD_STRIP



GL_POLYGON

- **Wszystkie wierzchołki wielokąta mają leżeć w jednej płaszczyźnie**
- **Nie wolno definiować wielokątów wklęsłych**

Kierunek rysowania prymitywów zamkniętych



Kierunek tworzenia obiektu
odwrotny do obiegu wskazówek
zegara (ang. CCW-Counterclockwise)

Kierunek tworzenia obiektu
zgodny z kierunkiem obiegu wskazówek
zegara (ang. CW-Clockwise)

```
glFrontFace(GL_CW);  //zgodnie ze wskazówkami zegara

glFrontFace(GL_CCW); // przeciwnie do wskazówek
```

Tworzenie zamkniętych brył

```
glBegin(GL_TRIANGLE_FAN);
glVertex3f(0.0f, 0.0f, 75.0f);
for(angle=0.0f; angle<(2.0f*3.15); angle+=(3.15/8.0f))
{   x = 50.0f*sin(angle);
    y = 50.0f*cos(angle);
    if((iPivot %2) == 0)    glColor3f(0.0f, 1.0f, 0.0f);
    else                    glColor3f(1.0f, 0.0f, 0.0f);
    iPivot++;
    glVertex2f(x, y);} glEnd();
glBegin(GL_TRIANGLE_FAN);
glVertex2f(0.0f, 0.0f);
for(angle=0.0f; angle<(2.0f*3.15); angle+=(3.15/8.0f))
{   x = 50.0f*sin(angle);
    y = 50.0f*cos(angle);
    if((iPivot %2) == 0)    glColor3f(0.0f, 1.0f, 0.0f);
    else                    glColor3f(1.0f, 0.0f, 0.0f);
    iPivot++;
    glVertex2f(x, y);} glEnd();
```

Tworzenie zamkniętych brył

- **Interpretacja kolorów wierzchołków:**

```
glShadeModel(GL_FLAT);        // glShadeModel(GL_SMOOTH);
```

- **Bufor głębokości:**

```
glEnable(GL_DEPTH_TEST);    // glDisable(GL_DEPTH_TEST);
```

- **Wyłączanie wyświetlania wybranych powierzchni:**

```
glEnable(GL_CULL_FACE);     // gDisable(GL_CULL_FACE);
```

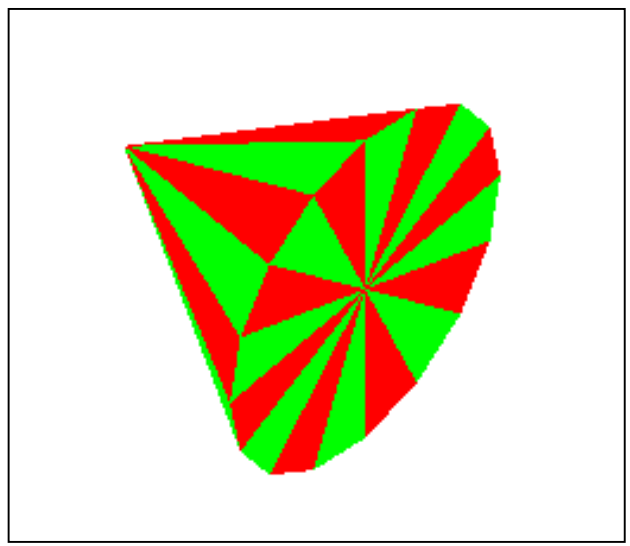
- **Ustalenie sposobu wyświetlania stron wielokąta:**

```
glPolygonMode(GL_BACK,GL_LINE);// glPolygonMode(GL_BACK,GL_FILL);
```

- **Dostepne parametry wywołania glPolygonMode:**

```
glPolygonMode(face,           mode);
```

GL_FRONT	GL_POINT
GL_FRONT	GL_LINE
GL_FRONT_AND_BACK	GL_FILL



Funkcja glVertex - charakterystyka

- **Dostępne funkcje rysowania wierzchołków:**

```
glVertex2d, glVertex2f, glVertex2i, glVertex2s,
glVertex3d, glVertex3f, glVertex3i, glVertex3s,
glVertex4d, glVertex4f, glVertex4i, glVertex4s,
glVertex2dv, glVertex2fv, glVertex2iv, glVertex2sv,
glVertex3dv, glVertex3fv, glVertex3iv, glVertex3sv, glVertex4dv,
glVertex4fv, glVertex4iv, glVertex4sv
```

- **Interpretacja oznaczeń:**

```
d -    double (GLdouble)
f -    float (GLfloat)
i -    int (GLint)
s -    short (GLshort)
2 -    podawane są współrzędne x i y
3 -    podawane są współrzędne x, y i z
4 -    podawane są współrzędne x, y, z i w = 1.0 (domyślnie),
        prawdziwe współrzędne oblicza się jako x/w, y/w, z/w
v -    parametrem wywołania funkcji jest tablica zawierające
       odpowiednio 2, 3 lub 4 dane określające współrzędne
       wierzchołka
```

Funkcja glVertex - przykłady

```
{ double x=10.0, y=0.0, z=5.0, w=1.0;
  glBegin(GL_POINTS);
       glVertex4d(x,y,z,w);
  glEnd();}

//-------------------------------------------------------------
{ int v_tab1[3] ={20,20,0};
  int v_tab2[3] ={-20,20,0};
  int v_tab3[3] ={-20,-20,0};
  int v_tab4[3] ={20,-20,0};
  glBegin(GL_QUADS);
       glVertex3iv(v_tab1);        glVertex3iv(v_tab2);
       glVertex3iv(v_tab3);        glVertex3iv(v_tab4);
  glEnd();}

//-------------------------------------------------------------
{ int i;
 int v_tab[4][3] =  {{20,20,0},{-20,20,0},{-20,-20,0},{20,-20,0}
                    };
 glBegin(GL_QUADS);
 for(i=0;i<4;i++)   glVertex3iv(v_tab[i]);
glEnd();}
```

Funkcje glBegin / glEnd – zasady stosowania

- **Dostępne prymitywy graficzne:**

```
GL_POINTS
GL_LINES
GL_LINE_STRIP
GL_LINE_LOOP
GL_TRIANGLES
GL_TRIANGLE_STRIP
GL_TRIANGLE_FAN
GL_QUADS
GL_QUAD_STRIP
GL_POLYGON
```

- **Podzbiór funkcji OpenGL, które można wywołać pomiędzy glBegin a glEnd:**

```
glVertex
glColor
glIndex
glNormal
glTexCoord
glEvalCoord
glEvalPoint
glMaterial
glEdgeFlag
```

Zasady konstruowania typowej sceny

1. **Przygotowanie siatek**
   1. **Siatki skonstruować można z zastosowaniem programów graficznych lub bezpośrednio definiując prymitywy graficzne i położenie ich wierzchołków.**
   2. **Siatki zamkniętych brył warto zbudować z „zewnętrznych" ścian (można wyłączyć wtedy liczenie ścian wewnętrznych – przyspieszenie obliczeń). Do tworzenia brył należy zastosować wielokąty a nie linie, czy łamane. Przy czym w pierwszym etapie można wymusić wyświetlanie wielokątów w postaci siatek (większa czytelność).**
   3. **Każdy element sceny, który będzie się poruszał w stosunku do innych elementów należy zdefiniować jako osobną siatkę.**
   4. **W tworzeniu siatek należy się posługiwać lokalnym układem współrzędnych, nie należy definiować położenia siatki (obiektu) na scenie.**
   5. **Sąsiednie ściany brył, które współdzielą „ostrą" krawędź należy wykonać z zastosowaniem oddzielnych prymitywów.**
   6. **Poszczególne elementy siatki zamknąć w jednostka programowaych (język C – funkcje, język C++ - klasy).**
   7. **Siatki do wielokrotnego użycia (np. walec) zdefiniować w postaci sparametryzowanej (np. Funkcje z parametrami wywołania);**

Zasady konstruowania typowej sceny

2. Zdefiniowanie transformacji przestrzennych i położenia kamery:
   1. Powiązanie ruchomych elementów poszczególnych obiektów za pomocą lokalnych transformacji przestrzennych;
   2. Ustalenie zasad ruchu obiektów i kamery na scenie.
3. Zdefiniowanie oświetlenia sceny:
   1. Włączenie oświetlenia na scenie;
   2. Zdefiniowanie normalnych do ścian obiektów, które będą oświetlane;
   3. Dla elementów nieoświetlanych dynamicznie wyłączać oświetlenie.
4. Teksturowanie:
   1. Przekształcić pliki graficzne w tekstury OpenGL;
   2. Powiązać współrzędne tekstur ze współrzędnymi wierzchołków.
5. Zdefiniowanie interfejsu użytkownika:
   1. Zaproponować zestaw klawiszy lub okien i elementów menu, które będą służyć do wpływania przez użytkownika na elementy sceny;
   2. Zastosować możliwość „dotykania” obiektów sceny:
6. Dodatkowe elementy – mgła, przezroczystość, mapy oświetlenia itp..

Przykład – definiowanie siatki walca

```
void walec(double r, double h)
{ double angle,x,y,z;
  glBegin(GL_TRIANGLE_FAN);
  glVertex3f(0.0f, 0.0f, 0.0f);
  for(angle = 0.0f; angle < (2.0*3.15); angle += (3.15/8.0f))
       { x = r*sin(angle);
         y = r*cos(angle);
         glVertex2d(x, y);}
  glEnd();
  glBegin(GL_QUAD_STRIP);
  for(angle = 0.0f; angle < (2.0*3.15); angle += (3.15/8.0f))
       { x = r*sin(angle);
         y = r*cos(angle);
         z = h;
         glVertex3d(x, y, 0);
         glVertex3d(x, y, z);}
  glEnd();}
void RenderScene(void)
{ //...
  glColor3d(1.0,0.0,0.0);
  glPolygonMode(GL_BACK,GL_LINE);
  walec(40,20); //...
}
```